sc2 Hardware Library Reference Manual

Jan Frigo
Los Alamos National Laboratory
MSD440
Los Alamos, NM 87545
jfrigo@lanl.gov

Contents

List of Figures

1. Introduction

sc2 is a new implementation of the Streams-C language and compiler. The Streams-C programming model is that of communicating processes. A system consists of a collection of *processes* that communicate using *streams* and *signals*. Processes can run either in software on conventional processors (SP) or in hardware on FPGA processors (HP). The sc2 compiler is used to compile FPGA processes in hardware. The compiler translates a subset of C into Register-Transfer-Level (RTL) VHDL that is synthesizable on FPGAs. The sc2 compiler synthesizes hardware circuits for one or more FPGAs as well as a set of communicating processes on conventional processors. The language extensions allow pipelined stream computation, so that the generated hardware/software is capable of pipelining a computation across multiple FPGAs and the conventional processor.

Our programming model is targeted at stream-oriented FPGA applications. Characteristics of stream-oriented computing include high-data-rate flow of one or more data sources, fixed size, small stream payload (one byte to one word), compute-intensive operations, usually low precision fixed point on the data stream, access to small local memories holding coefficients and other constants, and occasional synchronization between computational phases.

This sc2 release provides hardware libraries for the Annapolis Micro Systems (AMS) Firebird board, which contains one Xilinx Virtex-E FPGA on a 64-bit PCI bus. This manual describes the Streams-C hardware library used to facilitate the hardware interface between the Streams-C hardware processes and the target hardware board. It describes how to compile a VHDL simulation of the hardware processes and how to generate an executable (bit stream) for the hardware.

1.1. Scope of this Reference Manual

The purpose of this reference manual is to enable the user to generate a bit stream for hardware and describe the hardware interface between the sc2 generated hardware processes and the AMS Firebird board. It is assumed the user is familiar with VHDL or other hardware description languages.

2. Streams-C Hardware Library Components

- Hardware Streams Components - high bandwidth synchronous communications

  StrmFifoWrite - software to hardware fifo module

  StrmFifoRead - hardware to software fifo module

  StrmIntraRead - hardware to hardware streams fifo module

  StrmIntraWrite - hardware to hardware streams fifo module

  Fifo16 - internal fifo used in the stream fifo modules.

  Fifo32 - internal fifo used in the stream fifo modules.

  Fifo64 - internal fifo used in the stream fifo modules.

- Hardware Signal Components - low bandwidth asynchronous communications

  Sig_Recv - software to hardware process signal receive

  Sig_Send - hardware to software process signal send

- External Memory Component - 32-bit and 64-bit memories

  Sc_Mem - interface from hardware process to the 64 bit memories

  Sc_Mem4 - interface from hardware process to the 32-bit external memory

  (I) On our Firebird boards (Annapolis MicroSystems Firebird board pn# 12676-0000 Rev A) we have experienced loss of data integrity when more than 500k words are read/written to external 32-bit memory, mem 4.

- Block Ram Components - block ram memory for all Streams-C data types

  Sc_Bram - block rams connected to hardware processes

  Sc_Dpram - 64-bit by 256 deep dual port ram VHDL module. One port is connected to a software process via the LAD bus and the other to a hardware process.

- Pipeline Control Components

  Indefinite - controls pipeline for indefinite loops (*while*)

  Definite - controls pipeline for definite loops (*for*)

3. Functional Streams-C Hardware Library Description

3.1. Process Component

The process module has two main components, a datapath process and a sequencer. The datapath process entity consists of a datapath entity and a pipeline control entity (if a *while* or *for* loop is pipelined within the process). The sequencer is a state machine for sequencing through the instruction set of the process module. If a stream, signal, external memory, or blockram is used in the process, the signal interface for these components is included on the process module's port. See figure 1.

3.2. Stream Components

The hardware stream component is used for high bandwidth, synchronous communication between the processes. These are parameterized modules with respect to data register width and fifo depth. Data width can be any Streams-C data types. The fifo depths currently implemented are 16, 32, and 64. The modules have separate Data, Enable and Ready signals as shown in figure 2 the Enable is an input which means the data on the input or output port is valid. The Ready is an output signal indicating the module is ready to receive data.

(I) All the Streams-C data types are supported in the hardware library and can be used for hardware-to-hardware streams or signal connections. For hardware-to-software streams connections, 32-bit and 64-bit data types are supported. For hardware-to-software signal connections, data types less than or equal to 64-bits are supported.

3.3. Signal and Parameter Components

The hardware signal component is used for low bandwidth, asynchronous coordination or flow control between processes. Signals are parameterizable with respect to data width. The convention for Enable, Ready, and Data is shown in figure 2. The Enable is an input to the module, SigSend, which means the data on the input port is valid. The Ready is an output signal indicating the module, SigRecv, has valid data on the output port.

The hardware parameter component is used for passing a variable between processes and to initiate a hardware process. Parameters are parameterizable with respect to data width. The modules have separate Data and Enable signals. The Enable is an output that means the data on the output port is valid. A hardware parameter is implemented as an input signal by the sc2 compiler.

4. Memory Interfaces

4.1. External Memory

When Memory is invoked in a hardware process, if a pragma does not exist to indicate a memory type, the compiler default is 64-bit external memory (Mem_0). When memory is used, the ports for address, data, and enables - MAR, MDR_IN, MDR_OUT, RD_EN, WR_EN respectivley appear on the hardware process entity. They are connected through the hardware streams library to the Firebird board. The hardware processes and streams components for external memory, SC_Mem (64-bit memories), SC_Mem4 (32-bit memory) connect to the Annapolis hardware with a Mem64_Mux_Priority_IF component. This Mem64_Mux allows multiple connections to external memory - one to connect to the LAD bus, allowing memory to be loaded from the host, and one or more as needed to connect to hardware processes. The Mem64_Mux_Priority_IF component connects to the LAD64_Mem64_Bridge component which facilitates loading to external memory via the LAD bus/PCI bus. Dual port rams (64-bit x 256 words deep) are contained within the LAD64_Mem64_Bridge component (each Bridge component uses 2 blocks of the Virtex Block RAM). The LAD64_Mux_IF is the mux interface to the LAD bus which connects to the PCI controller and the host as shown in figure 3. The 32-bit external memory, mem4, uses a separate set of Annapolis components - SC_Mem4, Mem32_Mux_Priority_IF, LAD64_Mem32_Bridge, LAD64_Mux_IF. Figure 4 describes the external memory types. Also reference /streamsc/apps/arch/Firebird.def

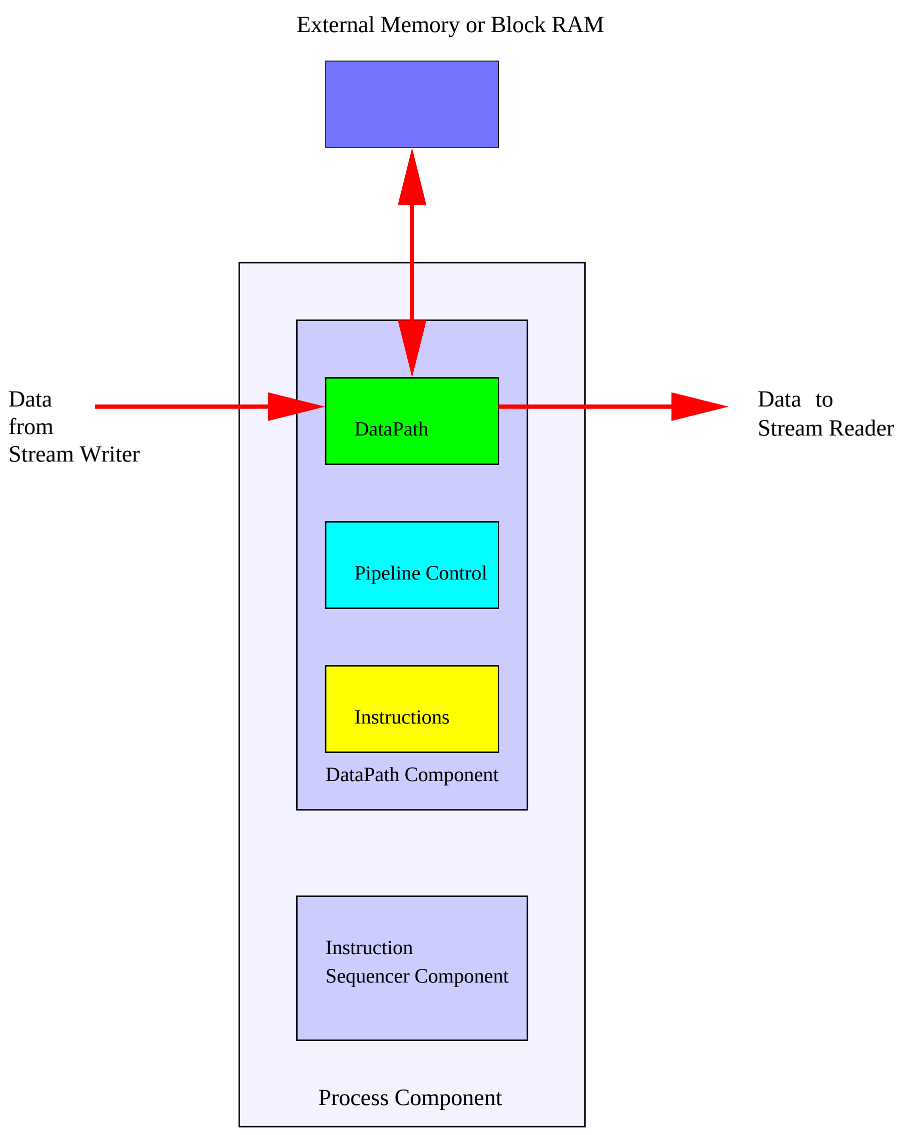

Figure 1. Hardware Process Component

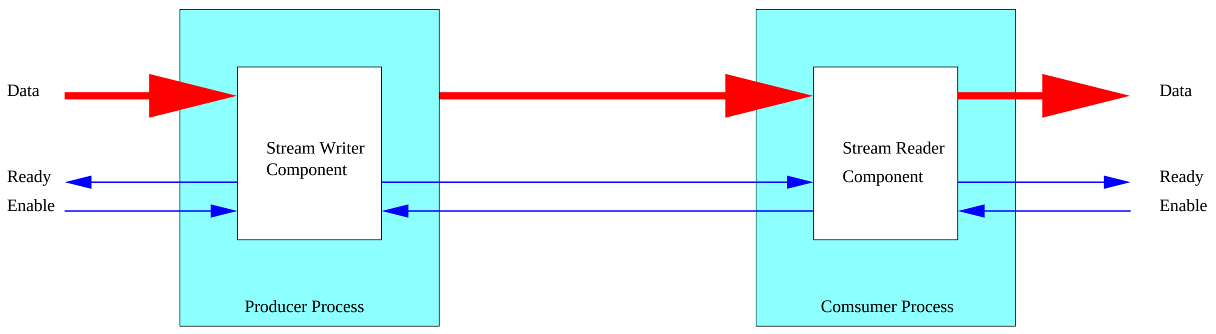

Figure 2. Hardware Streams Component

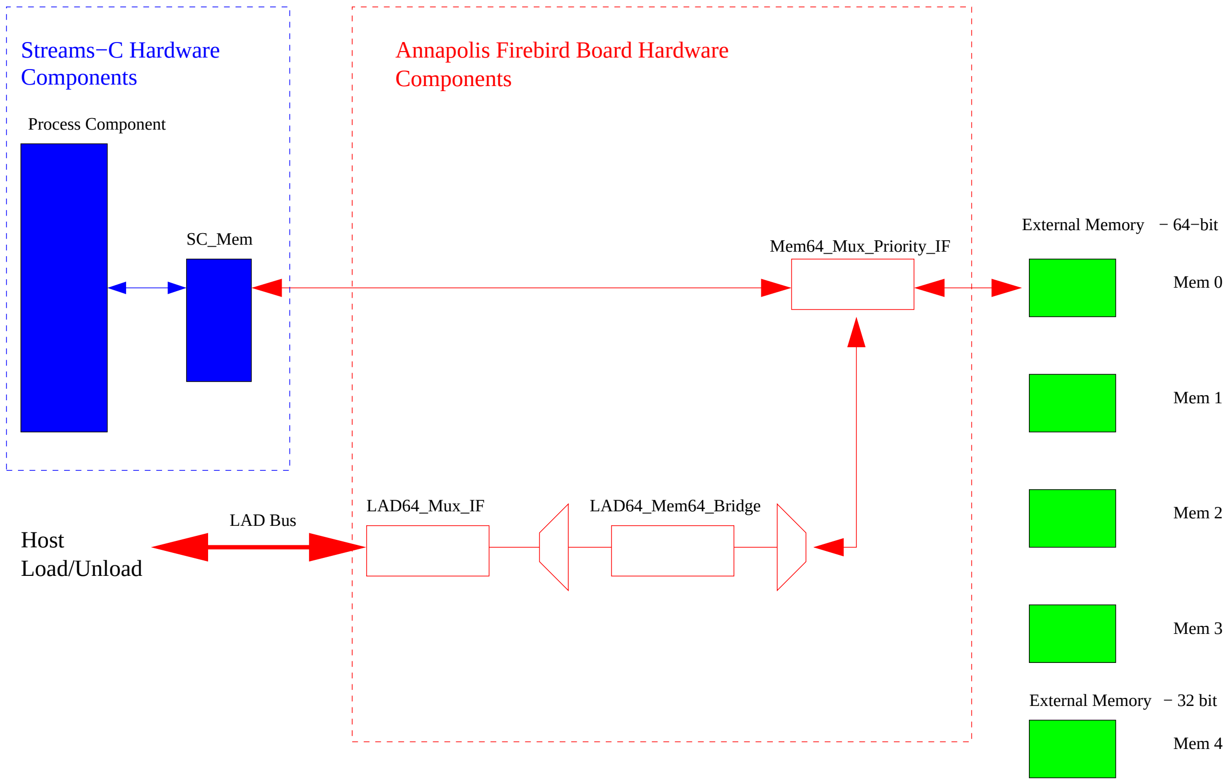

Figure 3. Streams-C External Memory Interface

Note: The Streams-C examples tested use a LAD bus speed of 66 MHz per the Annapolis PCI controller version 2.8. The LAD bus is adjustable to either 33MHz or 66MHz with PCI controller version 3.0.

4.2. Block RAM

For local data storage, the Streams-C hardware libraries allow the user to connect to block RAM with a stream or a host software process. See figures 6 and 7 for the Streams-C hardware implementation of block RAM.
Streams-C uses a pragma statement for selection of a block RAM type. Figure 5 shows the types of block RAM the Streams-C hardware library provides. (These type definitions are located in /streamsc/apps/arch/Firebird.def) The pragma statement for a block ram type, B_K_, of size 64-bit x 256 is shown below where A is the name of the array in the user program and the maximum size of A is 256. A read from block RAM can be placed in a local variable or in an output stream. These statements are identical to those used for external memory. A write and read with respect to block RAM is equivalent to the following C statements:

```
sc_uint32 A[256];
#pragma SC memory B_K_0 A
data = sc_stream_read(input_stream);
A[i] = data; //write data to block ram
data = A[i]; //read from block ram and place in a variable, data
for(i=i-1; i>=0; i--)
   sc_stream_write(output_stream, A[i]);//read from block ram and place in output_stream
```

The Sc_Dpram (64-bit x 256), hardware streams library component provides data transfer from the host to block RAM and vise versa. This library component uses dual ported RAM with one port connected via the LAD bus to the host and the other port connected to the hardware process port. For an example of how to load and unload block RAM from the host software process, reference the streamsc/apps/bram1 example.

type	width (bits)	size
mem_0	64	1000000
mem_1	64	1000000
mem_2	64	1000000
mem_3	64	1000000
mem_4	32	500000

Figure 4. Streams-C External Memory Types

Single Port type	width (bits)	size
B_A_	8	128
B_B_	8	256
B_C_	8	512
B_D_	16	128
B_E_	16	256
B_F_	16	512
B_G_	32	128
B_H_	32	256
B_I_	32	512
B_J_	64	128
B_K_	64	256
B_L_	64	512
B_M_	1	4096
B_N_	2	2048
B_O_	4	1024

Dual Port type	width (bits)	size
DP_2	64	256

Figure 5. Streams-C Block RAM Types

Streams-C expresses memory latency of a read(load) as the delay after the read is issued, until the data is available on the port's data-register. For example, a one-cycle latency means the data is available the cycle after the read is issued:

Latency one:

tick 1 - set mar and enable bit

tick 2 - copy mdr into user register.

With this definition, latency is always at least one. For the user who wishes to define a new hardware target board, read(load) latency is the delay required for the data to be available in the port's data-register. For the AMS Firebird board, an external memory write executes in one cycle, a read executes in 7 cycles. A memory optimization currently in the compiler stalls the datapath process when a read(load) is in progress. Thus, the latency is 1 cycle for external memory read (load) in the Firebird.def file. The Virtex-E chip executes a write and a read in one cycle using block RAM. The output from block ram is registered so the read latency is currently two cycles. See /streamsc/apps/arch/Firebird_mem.def for definitions of load and store latencies.

(I) The first column of the streamsc/apps/arch/Firebird.def file defines the number of types that are allocated. This number must be greater than one. For example,

1 BLOCK_RAM1 Memory B_A_ size 128

is an erroneous statement because the name, B_A_, causes a syntax error in gen_vhdl when only 1 block ram of type B_A_ is allocated in the Firebird.def file. The correct statement is

2 BLOCK_RAM1 Memory B_A_ size 128

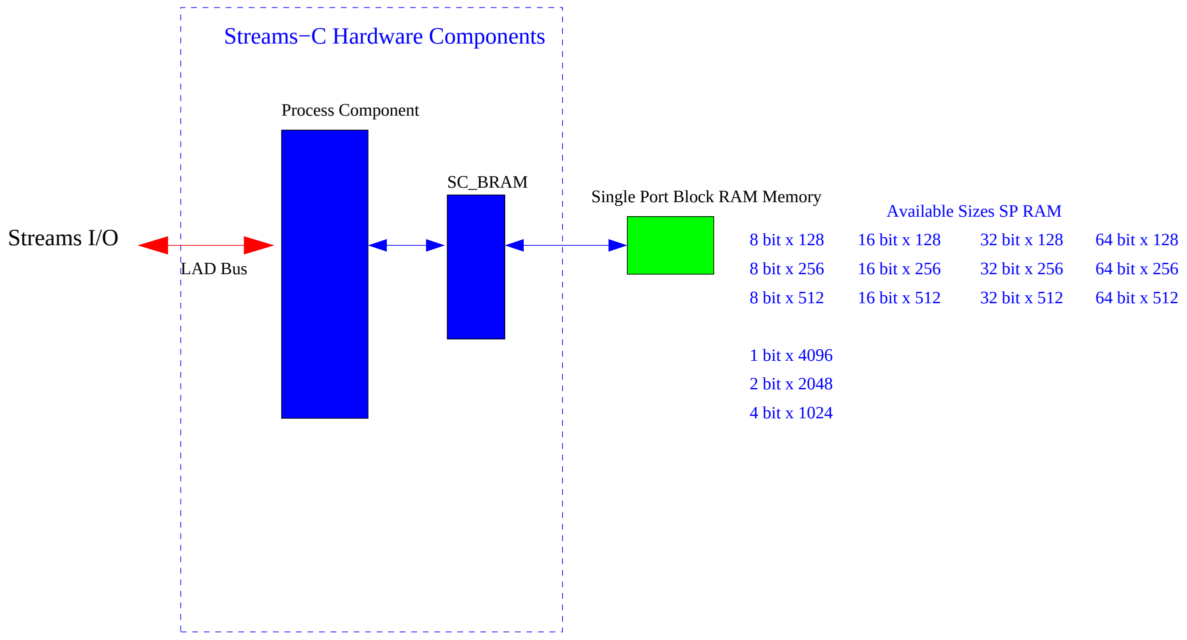

Figure 6. Streams-C Block RAM Memory Interface

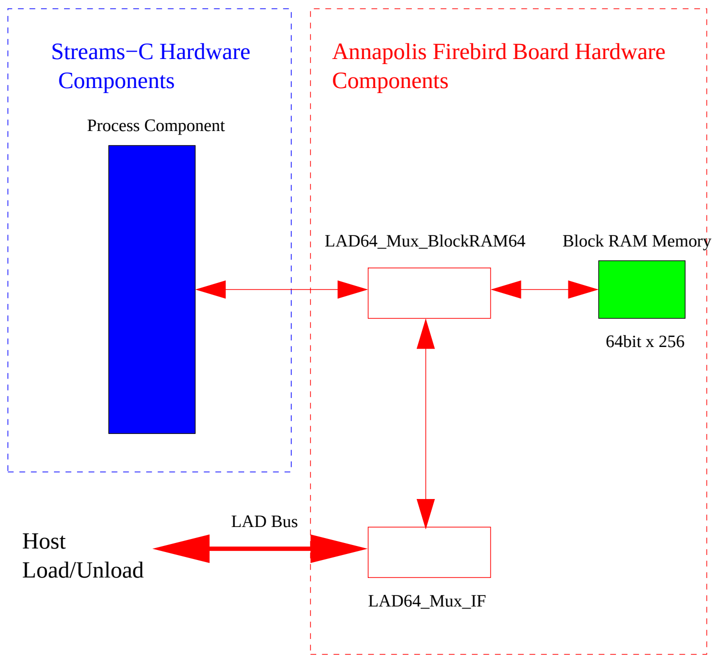

Figure 7. Streams-C Dual Port Block RAM Memory Interface

5. Using sc2

The generated VHDL command, ”make filename_all.vhd” generates the following VHDL output files: filename_all.vhd, filename_arch.vhd. The filename_all.vhd file contains the generated VHDL for the hardware processes which consists of a datapath, sequencer, and indefinite or definite hardware library component (if a *for* or *while* loop exists) for each hardware process defined in the Streams-C program, filename.sc The architecture file, filename_arch.vhd, uses the Streams-C hardware library to connect the hardware processes and any external memory or block ram to the target hardware board. This file is specific to AMS Firebird board, but can be modified to target a user defined board. (see section 8).

The following sections describe how to compile a VHDL simulation using ModelSim PE 5.5e, how to compile the VHDL project files for synthesis, and how to generate a hardware bit stream.

5.1. Tools Required

Modelsim PE 5.5e
Xilinx Core Gen 4.1i
Xilinx 5.2i Tools
Synplify 7.1.1
Annapolis VDHL version 3.2
Annapolis host API, version 5.0.0

5.2. VHDL simulation

- *Generate the Xilinx Core Libraries*

  Prior to compiling for Modelsim the Xilinx core library must be generated using Xilinx Core Gen 4.1i tool. How do I compile Xilinx Core Libraries on Modelsim? Go to the Xilinx website, Technical Answer Database: 8066 and Technical Answer Database: 2561

  http://support.xilinx.com/xlnx/xil_ans_display.jsp?iLanguageID=1&iCountryID=1&getPagePath=8066

  http://support.xilinx.com/xlnx/xil_ans_display.jsp?iLanguageID=1&iCountryID=1&getPagePath=2561

  Read these two Xilinx Techincal Answers carefully. Briefly, the compilation involves setting the correct paths to the library, setting some environment variables and compiling a vcom.do file. The compilation generates the behavioral models for the Xilinx Cores used with the Modelsim simulator. The /streamsc/vhdl_lib/sc_xilinx/ directory includes the vcom.do file needed to generate the libraries. Be sure you are using Xilinx Core Gen 4.1i.

  Note: If you are currently a Xilinx technology user, please contact the author for the pre-compiled Xilinx core libraries and the streamsc/vhdl_lib/sc_xilinx/ directory that contains the hardware library cores.

- *Set Library and Project Paths*

  Once the core libraries are compiled, edit the mti_vcom.do file to set paths to the Xilinx Core libraries, Annapolis hardware libraries, Streams-C hardware library, and your project directory. Make sure you have the Annapolis Board VHDL models installed in a local directory.

  ```
  set MODEL_TECH ”D:/ProgramFiles/modeltech”

  set ANNAPOLIS_BASE ”D:/Annapolis”

  set PROJECT_BASE ”F:/Firebird/strm/project”

  set SC_LIBRARY_BASE ”F:/streamsc/vhdl_lib”

  – Xilinx Logiblox support for RAM blocks

  vmap xilinxcorelib ”d:/xilinx/vhdl/src/XilinxCoreLib/xilinxcorelib”
  ```

  Place the Xilinx .mif files located in streamsc/vhdl_lib/sc_xilinx/sim in your PROJECT_BASE directory.

- *Edit your Project filename*

  ```
  vcom -93 -explicit -work PE0_Lib $PROJECT_BASE/filename_all.vhd

  vcom -93 -explicit -work PE0_Lib $PROJECT_BASE/filename_arch.vhd
  ```

- *Create your Host VHDL program*

  There are some example host programs and system configuration files in streamsc/apps/app_name/sim. These will show how to simulate a write/read to/from streams, signals, parameters, external memory and block ram for Modelsim PE. Construct a host program named, host_filename_arch.vhd, and edit the mti_vcom.do file line to include it in the simulation.

  Edit the system_cfg.vhd file to include the proper host architecture, Streams-C architecture configuration and external memory models.

  ```
  vcom -93 -explicit -work system $PROJECT_BASE/host_filename_arch.vhd
  vcom -93 -explicit -work system $PROJECT_BASE/system_cfg.vhd
  ```

- *Compile - Annapolis board libraries, Streams-C hardware libraries and Streams-C generated project files*

  mti_vcom.do

  Note: The following Warnings occur during modelsim compilation because the behavioral cores are not bound to the component until the design is loaded in the next step.

  ```
  # WARNING[1]: F:/Firebird/vhdl_lib/fifo16.vhd(110): No default binding for component: ”distram16x1”. (No entity named ”distram16x1” was found)
  # WARNING[1]: F:/Firebird/vhdl_lib/fifo16.vhd(110): No default binding for component: ”distram16x2”. (No entity named ”distram16x2” was found)
  etc...
  ```

- *Load system configuration*

  Once your project has compiled, go to the ModelSim *System* library and load system_cfg, the top level configuration module.

  Note: The Annapolis libraries only need to be compliled once for each project directory, so after a successful compile comment out the line #do $FIREBIRD_PCI_BASE/vhdl/system_vcom.do

5.3. VHDL synthesis

- *Set environment variables and edit project filenames*

  The VHDL synthesis tool used with Streams-C is Synplify 7.0. The vhdl_lib/sc_xilinx/syn directory has a synthesis project file, pe0.prj, which sets all the defaults needed for the synthesis tool. Edit the location of the paths in the pe0.prj file and edit the names of your project files

  ```
  –environment variables
  set PROJECT_BASE ”F:/firebird/strm/project”
  set ANNAPOLIS_BASE ”D:annapolis”
  set SC2_LIB_BASE ”F:/streamsc/vhdl_lib”
  –project files
  add_file -vhdl -lib PE0_Lib ”$PROJECT_BASE/filename_all.vhd”
  add_file -vhdl -lib PE0_Lib ”$PROJECT_BASE/filename_arch.vhd”
  ```

- *Set Clock frequency*

  Set the Clock frequency option to a desired design speed.

  ```
  set_option -frequency 66.000
  ```

- *Run the pe0.prj project in Synplify to generate a pe0.edf file*

5.4. Hardware bit stream generation

- *Copy your pe0.edf file to $PROJECT_BASE/pnr directory.*

  where $PROJECT_BASE is the location of your project files.

- *Set path for Annapolis VHDL models.*

  The /streamsc/vhdl_lib/sc_xilinx/pnr directory contains a makefile file. Edit this makefile to set the path for the location of the Annapolis VHDL models.

  set ANNAPOLIS_BASE "D:/Annapolis"

- *Place and Route the design*

  Place the Xilinx .edn files located in streamsc/vhdl_lib/sc_xilinx/pnr in your PROJECT_BASE directory.

  Open a cygwin bash shell or command prompt, and cd to your project directory then type, make pe0.x86

  where $ANNAPOLIS_BASE is the location of the Annapolis VHDL models. This command creates the bit stream - places and routes the hardware design. This step will take quite a bit of time depending on the size of the project. Decrease the M_CLK user constraints if faster routing time is desired. The user constraints file for place and route, $ANNAPOLIS_BASE/firebird_pci/template/syn/pe0/pe0_timing.ucf sets the defaults as follows:

  M_CLK 10ns

  K_CLK 15ns

  You have created a pe0.x86 binary file, the hardware bit stream. Use this bit stream with the runtime libraries to run the application on the hardware.

  The pe0.par log files from Xilinx reports the maximum frequency achieved during place and route. The user should set SC_MCLK, the desired frequency for running the bit stream on hardware, by setting defines in the filename.sc program:

  /// SOFTWARE_INCLUDE

  #define SC_MCLK 50.0

  /// SOFTWARE_INCLUDE_END

  If the defines above are not set by the user, the defaults for SC_MCLK and SC_UCLK are set to 20 MHz by the preprocessor. The Streams-C hardware components use the SC_MCLK for the user design speed. The SC_KCLK for the LAD Bus speed is set to 66 MHz by the sim_rt libraries. SC_UCLK is not currently being used.

6. Hardware Implementation Notes

- Be sure to check the version of the simulation, synthesis and place and route tools you are using with Streams-C. Streams-C was tested and developed with the following CAD tools:

  Modelsim PE 5.5e –VHDL simulation

  Xilinx Core Gen 4.1.03i –Xilinx behaviour models and cores for VHDL simulation and place and route

  Xilinx 5.2i Tools –bit stream, pe0.x86 file generation

  Synplify 7.1.1 –net list, pe0.edf file generation

  Annapolis VHDL version 3.2 –simulation, synthesis and place and route

  Annapolis host API, version 5.0.0 –the sim_rt libraries

  Note: Be sure to check the log files after place and route to determine the actual speed at which your design was routed. This value needs to be entered in the filename.sc file as described in section 5.4.

- The user may set SC_MCLK, the desired frequency for running the bit stream, by using a software include in the filename.sc program. A default is set to 20 MHz by the preprocessor if nothing is defined. The Streams-C hardware components use the SC_MCLK for the user design speed. the SC_KCLK for the LAD Bus speed is set to 66 MHz by the sim_rt libraries. SC_UCLK is not currently being used.

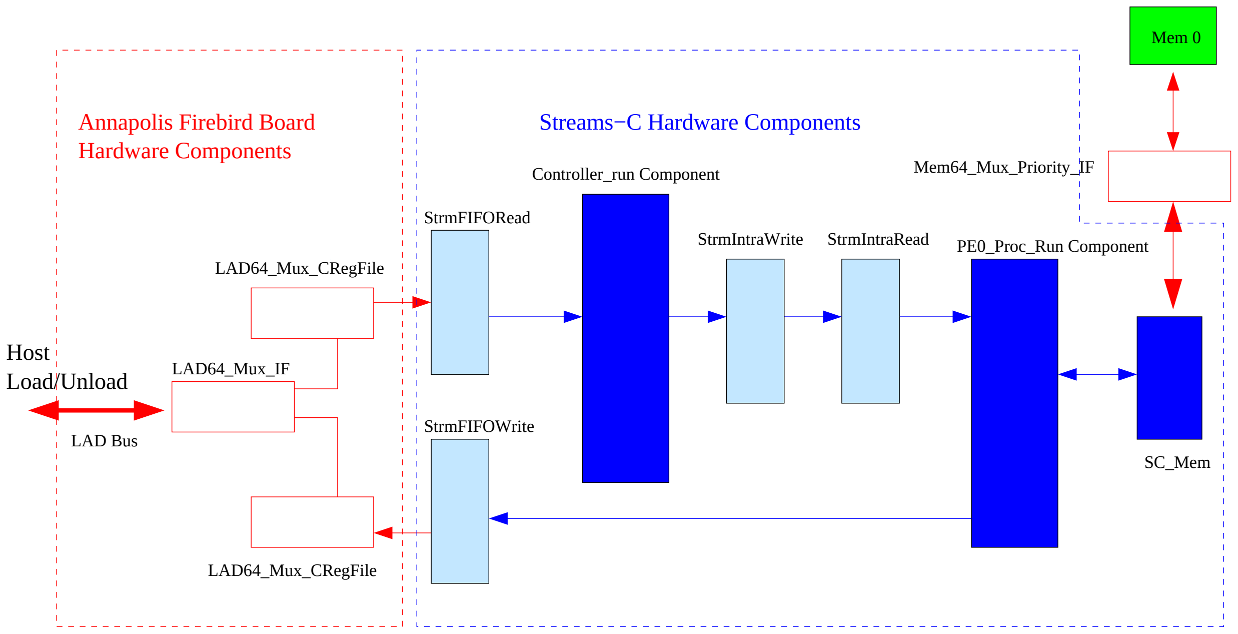

Figure 8. Example Architecture File

- Consider your application usage of streams and design speed issues. The LAD bus runs at 66 MHz on the Firebird board, so heavy demands on streams may result in slower overall system speed. Example apps/ppf1 yields a much faster run-time than apps/ppf.
- Use of multiple block rams in your user hardware processes may cause slower place and route design speeds due to the automatic placement with the Xilinx tools.
- Contact the author for the Xilinx specific hardware core libraries located at streamsc/vhdl_lib/sc_xilinx if you have a valid Xilinx license.

7. Examples

The program strm2.sc has two software processes and two hardware processes. The first software process host1, with run function host1_run opens an output stream and writes a sequence of integers to the stream. The bound on the loop ("iterations") is set by the input argument to the program invocation (eg. the invocation "strm2_sim 400" causes a sequence of integers from 0 to 399 to be written to the output stream).

The stream sent by host1 goes to a hardware process controller with run function controller_run. This process simply forwards the stream to the next hardware process, pe0_proc_run. pe0_proc_run has two phases. First it copies its input stream to memory. *Note: a pragma statement is not given for memory, therefore, the default is external 64-bit memory, mem 0.* When the whole stream has been read into memory, it reads back the data in reverse order and writes to its output stream. The final software process, host2, using run function host2_run, reads the stream from pe0_proc_run and prints out the data received from the stream.

The two hardware processes are generated by the sc2 compiler and placed in the strm2_all.vhd file. The hardware processes, controller_run, and pe0_proc_run are connected through the Hardware Streams Library components to the Annapolis board models in order to facilitate the hardware interface to the Firebird board. The strm2_arch.vhd file represented in figure 8 contains this "system connectivity".

The streamsc/vhdl_lib directory contains all of the Streams-C hardware library components. See section 3.2 for a description of the hardware library components.

8. Retarget Hardware Notes

This strm2_all.vhd file is designed to be “stand-alone” and can be retargeted to a different user system. The architecture file and the associated Streams-C hardware library modules for streams, signals, parameters and memory need to be redesigned for the user system. The Streams-C memory types in the streamsc/apps/arch/Firebird.def and Firebird_mem.def files need to be modified as well.

The process declarations and connects can reflect multiple FPGA’s via the naming convention, PE0, PE1, etc. in the PE.def file.

9. Hardware-Limitations

- CLB RAMs are not implemented in the Streams-C hardware libraries.
- Do not use external memory in an array of hardware processes. The Streams-C model does not arbitrate external memory access between multiple hareware processes. Use blockram types (not dual port ram or external memory types) in arrays of hardware processes.
- Do not use the same external memory (i.e. mem_0) in multiple hardware processes. The Streams-C model does not arbitrate external memory access between multiple hareware processes.
- Do not place an sc_wait() call within a pipelined loop.
- If a numeric value is posted with sc_post(), the value must be cast to the same type as the output signal.
- All the Streams-C data types are supported in the hardware library and can be used for hardware-to-hardware streams or signal connections. For hardware-to-software streams connections, 32-bit and 64-bit data types are supported. For hardware-to-software signal connections, data types less than or equal to 64-bits are supported.
- Multiplication and casting have been tested. See the *sc2 reference manual* section 4.0 and /streamsc/apps/mult1 and mult2 for examples.
- The sc2 compiler can pipeline blocks that contain control flow statements, i.e. if-else statements.
- The sc2 compiler can pipeline the inner-most loop(s) of a nested loop. See example /streamsc/apps/fastfold/fastfold.sc
- Unpipelined loops have not been exhaustively tested.
- For hardware synthesis divide is implemented only for powers of 2.
- The hardware implementation of the modulus intrinsic function, sc_mod(), only allows modulo by a power of 2.
- When using the unroll pragma, large unroll factors combined with large loop body may cause the scheduling phase of the compiler to run out of memory.
- The compiler will automatically pipeline all for and while loop even if a pragma is not used. To remove the always pipelining option for definite and indefinite loops, omit the -pipeline_all flag in streamsc/apps/Makefile
- For hardware synthesis the sc_catenate() bit intrinsic function requires unsigned arguments.
- For the hardware synthesis compiler, constant numbers (e.g. 100000) must be explicitly cast for use in bit intrinsic functions, defines, and assignments, otherwise their default type will be *signed int*.
- Hardware defines for constants must be sc data types or they must be explicitly cast in each instance they are used.
- For math operations the hardware synthesis compiler implements mixed type operations by casting the unsigned operand to signed.
- Frequent stalling will sometimes generate an error in unpipelined loops. We recommend use of external memory for inputting data from a software process to a hardware process (especially when using nested loops) in order to minimize stalling and improve the timing efficiency of the routed design.

	hw-to-hw connections	**sw-to-hw connections**
parameters	NA	1 - 64-bit types
signals	all types	1 - 64-bit types
streams	all types	32-bit, 64-bit types

Figure 9. Hardware Implementation of Streams-C Data Types

- Definite loops are implemented as indefinite loops in the hardware synthesis compiler. Use of multiple external memories within a pipelined loop can cause a dead-lock stall condition due to design constraints in the indefinite hardware controller module.

10. Style Issues

10.1. Optimization Hints

- Use functions sc_bit_insert() and sc_bit_extract() instead of the shift left or shift right (”<<” or ”>>”) operators.
- Pointers are not permitted. Indirect reference must be accomplished through array reference.
- If most of the 160 blocks of block RAMs are used for a memory intensive application, the overall timing of the design may decrease. The place and route tool routes these blocks automatically and timing could be affected.
- Use function sc_catenate() to concatenate multiple variables. See example /streamsc/apps/fastfold/fastfold.sc

10.2. Simulation vs. Synthesis

- Occasionally it is useful to write the code in one way for simulation and slightly different for synthesis. The IF_SIM macro is provided for this purpose.
- We have found that C compiler bugs sometimes cause large locally allocated arrays to get corrupted. Thus to circumvent the bug in simulation, the array is globally allocated during simulation and locally allocated for synthesis.
- The hardware synthesis compiler implements Streams-C data types for streams, signals and parameter connections per Figure 9.